

Apple® AppleTalk Data Stream Protocol Macintosh Operating System Driver Interface Preliminary Note

Apple Computer, Inc.

This manual and the software described in it are copyrighted, with all rights reserved. Under the copyright laws, this manual or the software may not be copied, in whole or in part, without written consent of Apple, except in the normal use of the software or to make a backup copy of the software. The same proprietary and copyright notices must be affixed to any permitted copies as were affixed to the original. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased (with all backup copies) may be sold, given, or loaned to another person. Under the law, copying includes translating into another language or format.

You may use the software on any computer owned by you, but extra copies cannot be made for this purpose.

© Apple Computer, Inc., 1988

20525 Mariani Avenue
Cupertino, CA 95014
(408) 996-1010

Simultaneously published in the United States and Canada.

Apple, the Apple logo, AppleTalk, and Macintosh are registered trademarks of Apple Computer, Inc.

ITC Garamond is a registered trademark of International Typeface Corporation.

Microsoft is a registered trademark of Microsoft Corporation.

PostScript is a registered trademark of Adobe Systems Incorporated.

Contents

Preface / v

1 About AppleTalk Data Stream Protocol / 1

Using ADSP / 2
- Connection ends / 2
- Connection control block / 3
- User routines / 4

2 ADSP routines / 5

Create Connection End / 6
Remove Connection End / 7
Open Connection / 8
Close Connection / 10
Create Connection Listener / 11
Remove Connection Listener / 12
Listen For Connection Request / 13
Deny Connection Request / 14
Get Status / 15
Read Bytes / 16
Write Bytes / 17
Send Attention Message / 18
Set Options / 19
Forward Reset / 21
Get New CID / 22
Example / 23

Appendix: Summary of ADSP data structures / 25

Constants / 26
Data types / 28
Assembly language information / 31

Preface

THIS PRELIMINARY NOTE describes the Macintosh® operating system driver interface to the AppleTalk® Data Stream Protocol (ADSP), which is a protocol that provides the client with a powerful interface to an AppleTalk network.

What's in this note

This note is divided into two chapters and an appendix that contain the following information:

- Chapter 1, "About AppleTalk Data Stream Protocol," explains concepts and components used in ADSP.
- Chapter 2, "ADSP Routines," describes each ADSP function call, including parameter blocks and result codes, and provides a sample client program.
- The Appendix, "Summary of ADSP Data Structures," presents ADSP constants, data types, and assembly language information.

Who should read this note

This note is intended for software developers writing applications that use ADSP services. To understand and use this document, developers should be familiar with:

- the Device Manager, described in Chapter 6 of *Inside Macintosh,* Volume II
- the AppleTalk Manager, described in Chapter 10 of *Inside Macintosh,* Volume II

Associated documents

The following documents supplement the information in this note:

- the *AppleTalk Data Stream Protocol Specification,* which explains ADSP in detail (this information will be incorporated into the book, *Inside AppleTalk*)
- *Inside AppleTalk,* which explains the AppleTalk protocols in detail

Chapter 1 About AppleTalk Data Stream Protocol

THE APPLETALK® DATA STREAM PROTOCOL (ADSP) is a symmetric, connection-oriented protocol that guarantees the ordered delivery of full-duplex streams of bytes between two given sockets in an AppleTalk internet. ADSP is a client of the Datagram Delivery Protocol (DDP) and a peer to the AppleTalk Session Protocol (ASP).

A connection is an association between two sockets that allows the reliable, full-duplex flow of data bytes between the sockets. A connection is composed of two connection ends, each of which is uniquely identified by its internet socket address and a connection identifier (also called a CID).

The client can use ADSP to create and remove connection ends, to request connections with remote ends, to wait passively for connection requests from remote ends, to read and write on open connections, and to close connections.

ADSP features include

- a built-in flow-control mechanism, ensuring that a sender never sends bytes to a receiver with no buffer space
- an end-of-message mechanism that enables the client to break streams of bytes into logical messages
- attention messages that allow clients to signal each other outside the normal flow of data
- a forward-reset mechanism that enables a sender to abort the delivery of all outstanding bytes sent to the remote client

Using ADSP

ADSP is implemented as a Macintosh® operating system driver. To use ADSP, the client must first initialize the AppleTalk Manager by opening the .MPP driver and then initialize ADSP by opening the .DSP driver.

▲ **Warning** The client should never attempt to close .MPP or .DSP because other processes may be using these drivers. ▲

The Device Manager handles communication between the client and the ADSP driver. The control routine in the Device Manager is used to access ADSP functions; the client passes a different *csCode* for each function.

ADSP provides a facility for building connection servers. A set of advanced calls enables a client to set up a connection listener to receive Open Connection requests from remote clients and to distribute these requests to other connection ends for further processing.

Connection ends

The memory ADSP requires to maintain a connection end is provided by the client. To create a connection end, the client must allocate a connection control block (CCB), a send queue, a receive queue, and an attention message buffer for ADSP's internal use. Pointers to the required memory are passed to ADSP in a control call with *csCode* set to *dspInit*. This call creates and initializes the connection end and returns a *refnum* that identifies the connection end in subsequent calls to ADSP.

◆ *Note:* The memory allocated and passed to ADSP to maintain a connection end becomes the exclusive property of ADSP for the life of the connection end. The client must ensure that the memory remains locked and unaltered until ADSP removes the connection end.

The connection end is initialized to a closed state. A control call with *csCode* set to *dspOpen* is used to open a connection with a remote connection end or to wait passively for a remote client to open a connection. The AppleTalk Name Binding Protocol (NBP) can be used to register a name for a connection end or connection server and to determine the addresses of other NBP-registered connections.

Once a connection is open, the client makes control calls to read data from or to write data to the remote connection end. When the client no longer requires the connection, the control call *dspClose* returns the connection end to a closed state, and *dspRemove* closes the connection and deletes the connection end from ADSP's internal data structures. The memory allocated for the connection end's associated data structures can be released only after the *dspRemove* call has completed.

Connection control block

The CCB is a block of memory, allocated by the client, that ADSP uses to keep state information about the connection end. Most of the CCB fields are for ADSP's internal use and are not visible to the client. The CCB must not be altered or moved in memory until ADSP is called to remove the connection end. The CCB fields are as follows:

Field	Description
ccbLink	pointer to next CCB (for the exclusive use of ADSP)
refnum	reference number of this CCB
state	current state of the connection end
userFlags	connection event flags for the client
localSocket	DDP socket number for reading from and writing to this connection end
remoteAddress	internet socket address of the remote connection end
attnCode	attention code of incoming attention message
attnSize	size of incoming attention message
attnPtr	pointer to buffer for incoming attention messages
reserved	additional fields (for the exclusive use of ADSP)

The client must never alter CCB fields (with the exception of *userFlags*). However, the client may poll CCB fields to determine the state of the connection.

The *refnum* field, a reference number for the connection end, is generated by ADSP when the connection end is created. This *refnum* is used by the client and ADSP to refer to the connection end.

The client may poll the value of *state* to determine the current state of the connection end. The possible states are listed in the Appendix, "Summary of ADSP Data Structures."

When an unsolicited event occurs on the connection end, ADSP sets an appropriate bit in the *userFlags* field of the CCB. The client tests the bits to determine what occurred. Events that can occur include receipt of an attention message, the receipt of a forward reset, the closing of the connection by the remote end, or the tearing down of the connection because the remote end has become unreachable. The client must clear the bits in the *userFlags* field after the corresponding event has been processed.

The *localSocket* field contains the DDP socket number through which the connection end transmits and receives packets. The *remoteAddress* field contains the internet socket address of the remote connection end (if any).

The *attnPtr* field points to the client buffer that is available to receive attention messages from the remote connection end. The *attnSize* field contains the size (in bytes) of any attention message received from the remote client (from 0 to 570 bytes). The *attnCode* field contains the client-definable code that was received.

User routines

When creating a connection end, the client may specify the address of a user routine to be called whenever an unsolicited connection event occurs. Unsolicited connection events are those that do not occur as a direct result of a client control call, for instance, the receipt of an attention message or close advice from the remote end, or the tearing down of the connection because of a timeout. The various events are listed in the Appendix, "Summary of ADSP Data Structures."

The user routine is called with register A1 pointing to the CCB of the connection end. By examining the fields of the CCB, the routine can determine the connection (from the *refnum* field), the event (by testing bits in the *userFlags* field), and the resulting state of the connection (from the *state* field). If an attention message is received, the routine can access the message from the *attnSize* and *attnPtr* fields.

◆ *Note:* The user routine is similar to an ioCompletion routine. It is called at interrupt level and must follow all rules regarding interrupt-level routines. Refer to the discussion of the Device Manager in *Inside Macintosh*, Volume II, for details. If the limitations of an interrupt-level routine are unsuitable for your application, use the alternate method of periodically polling the *userFlags* field of the CCB for connection events.

Chapter 2 ADSP routines

THIS CHAPTER describes each of the ADSP function calls and lists the fields in the parameter block affected by the call. It also identifies possible result codes.

The number next to each parameter name indicates the byte offset of the parameter from the start of the parameter block pointed to by register A0. An arrow next to each parameter name indicates whether it's an input, output, or input/output parameter:

Arrow	Meaning
—>	parameter is passed to the routine
<—	parameter is returned by the routine
<—>	parameter is passed to and returned by the routine

Create Connection End

This call creates and initializes a connection end. The *refnum* of the CCB is returned in the *ccbRefNum* parameter. The client can use this value to reference the connection end in subsequent calls to ADSP. The caller provides a pointer to a CCB in the parameter *ccbPtr*. The CCB becomes the exclusive property of ADSP and must not be moved or altered until the client calls ADSP to remove the connection end. The client passes to *userRoutine* the address of a routine to be called in case of a connection event.

An ADSP connection end requires buffer space to hold incoming and outgoing bytes. The send queue holds all bytes the client has asked ADSP to send over the connection whose delivery to the remove client has not yet been acknowledged. The receive queue buffers bytes received from the remote end until the client is able to read them. These queues are allocated by the client and become the exclusive property of ADSP; the queues must not be altered or moved until the connection end is removed. Pointers to the queues are passed in the fields *sendQueue* and *recvQueue*. The two queue size variables (*sendQSize*, *recvQSize*) are used to pass the size (in bytes) of these client-supplied buffers. When allocating the queues, the client should determine the size using the following rule:

- queue size = desired size + (desired size / 8) + 1

The *attnPtr* parameter contains the address of a client-supplied buffer that receives attention messages from the remote connection end. The attention buffer should be the constant *attnBufSize* bytes in size. The buffer must remain locked until the connection end is removed.

The *localSocket* parameter specifies the connection end's DDP socket. A value of 0 causes DDP to allocate a socket dynamically. The socket number is returned when the call completes.

Parameter block	—>	26	csCode	word	always dspInit
	<—	32	ccbRefNum	word	returns refnum assigned to connection end
	—>	34	ccbPtr	long	pointer to CCB
	—>	38	userRoutine	long	routine to call on connection events
	—>	42	sendQSize	word	size in bytes of the send queue
	—>	44	sendQueue	long	pointer to send queue
	—>	48	recvQSize	word	size in bytes of the receive queue
	—>	50	recvQueue	long	pointer to receive queue
	—>	54	attnPtr	long	pointer to buffer for incoming attention messages
	<—>	58	localSocket	byte	DDP socket number for this connection end
Result codes	ddpSktErr		error opening socket		

Remove Connection End

This call closes any open connection and removes the connection end specified by the parameter *ccbRefNum*. The DDP socket is closed if it is not in use by another connection end. Upon completion of this call, the CCB, attention buffer, and queue resources are no longer needed, and the memory can be released by the client. If the *abort* flag is nonzero, any outstanding client requests on the connection end are aborted, and all data in the send queue is discarded.

Parameter block	—>	26	csCode	word	always dspRemove
	—>	32	ccbRefNum	word	refnum of connection end
	—>	34	abort	byte	abort the connection flag
Result codes	errRefNum		bad connection refnum		

Open Connection

This call is used to set the opening state for a connection end. The state of the connection end must initially be closed. The connection end can be set into four opening states by setting the parameter *ocMode* to one of the following constants: *ocRequest, ocPassive, ocAccept, ocEstablish.*

Open mode *ocRequest* specifies that ADSP should initiate opening a connection with the remote address specified in the parameter *remoteAddress.* The *filterAddress* parameter is used to filter connection ends responding to the Open Connection request. A 0 in the network number, node identifier, or socket number of *filterAddress* means that a connection can be established with any connection end on any network, node, or socket, respectively. Setting *filterAddress* to be the same as *remoteAddress* means that a connection will be established only with a connection end on the specified *remoteAddress.* An *ocRequest* Open Connection call completes when a connection is established, when an Open Connection Denial is received from a remote end, or when the maximum retries have been exceeded.

Open mode *ocPassive* sets the connection end to a passive opening state. The connection end starts the connection-opening dialog when an Open Connection request is received from some remote connection end. The *filterAddress* can be used to specify the remote addresses with which the connection end is willing to establish a connection. A 0 in the network number, node identifier, or socket number of *filterAddress* means that a connection can be established with any connection end on any network, node, or socket, respectively. An *ocPassive* Open Connection call is not complete until a valid Open Connection request is received from a remote connection end. When an Open Connection request is received, the connection end enters the opening state and completes in the same manner as in the *ocRequest* mode.

The third open mode, *ocAccept,* is used by connection servers to complete an open-connection dialog, establishing a connection with the remote address from which an Open Connection request had been received by the server's connection listener. The *remoteAddress, remoteCID, sendSeq, sendWindow,* and *attnSendSeq* parameters should be filled in using the respective parameters returned from the connection listener's listen request (see "Listen For Connection Request" later in this chapter). An *ocAccept* Open Connection call completes in the same manner as in the *ocRequest* mode.

With the last mode, *ocEstablish,* ADSP considers the connection end established and the connection state open. This mode allows two peer clients to establish their respective connection ends based on connection-opening information that has been negotiated outside of ADSP. It is the client's responsibility to provide values for the parameters *remoteCID, remoteAddress, sendSeq, sendWindow, recvSeq, attnSendSeq,* and *attnRecvSeq.* Get New CID should be called to assign a unique *localCID* on the connection end prior to opening a connection in this fashion (see "Get New CID" later in this chapter). The *ocEstablish* Open Connection call completes immediately.

The parameter *ocInterval* specifies the interval between retransmission of Open Connection requests in 10-tick (1/6-second) increments. The client can set the interval to any value from 1 (1/6 second) to 180 (30 seconds) inclusive. The default value of 6 (1 second) is used if the parameter is 0. This parameter should be specified for Open Connection modes *ocRequest, ocPassive,* and *ocAccept.*

The *ocMaximum* parameter specifies the total number of times an Open Connection request is transmitted. Passing 0 for this parameter causes the default value of 3 to be used, but the client can set the maximum to any value from 1 (only one Open Connection request is transmitted) to 255 inclusive (continuously retransmit Open Connection requests until an acknowledgment or denial is received). This parameter should be specified for Open Connection modes *ocRequest, ocPassive,* and *ocAccept.*

Parameter block

—>	26	csCode	word	always dspOpen
—>	32	ccbRefNum	word	refnum of connection end
<—	34	localCID	word	connection identifier of this connection end
<—>	36	remoteCID	word	connection identifier of remote connection end
<—>	38	remoteAddress	long	internet address of remote connection end
—>	42	filterAddress	long	filter for incoming Open Connection requests
<—>	46	sendSeq	long	initial send sequence number to use
<—>	50	sendWindow	word	initial size of remote end's receive buffer
—>	52	recvSeq	long	initial receive sequence number to use
<—>	56	attnSendSeq	long	initial attention send sequence number
—>	60	attnRecvSeq	long	initial attention receive sequence number
—>	64	ocMode	byte	connection-opening mode
—>	65	ocInterval	byte	interval between Open Connection requests
—>	66	ocMaximum	byte	maximum retries of Open Connection request

Result codes

errRefNum	bad connection refnum
errState	connection end must be closed
errOpening	open connection attempt failed
errAborted	request aborted by a dspRemove or dspClose call

Close Connection

This call closes any open connection and sets the state of the connection end to closed. If the *abort* flag is nonzero, any outstanding client requests on the connection end are aborted, and all data in the send queue is discarded.

Parameter block	—>	26	csCode	word	always dspClose
	—>	32	ccbRefNum	word	refnum of connection end
	—>	34	abort	byte	abort the connection flag

Result codes	errRefNum	bad connection refnum

Create Connection Listener

This call creates and initializes a connection listener. The caller provides a pointer to a CCB in the parameter *ccbPtr*. The CCB is used by ADSP to maintain the connection listener. The *refnum* of the CCB is returned in the *ccbRefNum* parameter. The client can use this value to reference the listener in subsequent calls to ADSP.

The *localSocket* parameter specifies the DDP socket the connection listener will use. A value of 0 causes DDP to allocate a socket dynamically. The socket number is returned when the call completes.

Parameter block	—>	26	csCode	word	always dspCLInit
	<—	32	ccbRefNum	word	returns refnum assigned to connection listener
	—>	34	ccbPtr	long	pointer to CCB
	<—>	58	localSocket	byte	DDP socket number for this connection end

Result codes	ddpSktErr	error opening socket

Remove Connection Listener

This call closes the connection listener specified by *refnum*. If the *abort* flag is nonzero, any outstanding client requests (such as Deny Connection Request calls) are aborted. Upon completion of this call, the CCB is no longer needed, and the memory it occupied can be released.

Parameter block	—>	26	csCode	word	always dspCLRemove
	—>	32	ccbRefNum	word	refnum of connection listener
	—>	34	abort	byte	abort the connection listener flag

Result codes errRefNum bad connection refnum

Listen For Connection Request

Connection servers use this call to listen for connection requests. The caller specifies the *refnum* of the connection listener in the parameter *ccbRefNum*. The call completes when ADSP receives an Open Connection request, on the connection listener's socket, that satisfies the address requirements specified in the *filterAddress* parameter. The client must then determine what to do with the connection request. If a connection can be opened, the client must call Open Connection on an available connection end and set the *ocMode* parameter to *ocAccept*. The values returned in the parameters *remoteCID*, *remoteAddress*, *sendSeq*, *sendWindow*, and *attnSendSeq* from the completed listener call should be passed in the respective parameters of the Open Connection call. If the request cannot be honored, the client should advise the remote end by calling Deny Connection Request, specifying the listener's *ccbRefNum*.

Several listen requests can be posted to the connection listener, and each request can have a different filter address specification (if desired).

Parameter block

—>	26	csCode	word	always dspCLListen
—>	32	ccbRefNum	word	refnum of connection end
<—	36	remoteCID	word	connection identifier of remote connection end
<—	38	remoteAddress	long	internet address of remote connection end
—>	42	filterAddress	long	filter for incoming Open Connection requests
<—	46	sendSeq	long	initial send sequence number to use
<—	50	sendWindow	word	initial size of remote end's receive buffer
<—	56	attnSendSeq	long	initial attention send sequence number to use

Result codes

errRefNum	bad connection refnum
errState	not a connection listener
errAborted	request aborted by a dspRemove call

Deny Connection Request

This call is used by the client of a connection listener to advise a remote end that an Open Connection request cannot be honored. The caller should specify the connection listener in the parameter *ccbRefNum*. The *remoteCID* and *remoteAddress* parameters should be filled in using the corresponding values returned from the completed listener call.

Parameter block	—>	26	csCode	word	always dspCLDeny
	—>	32	ccbRefNum	word	refnum of connection listener
	—>	36	remoteCID	word	connection identifier of remote connection end
	—>	38	remoteAddress	long	internet address of remote connection end

Result codes	errRefNum	bad connection refnum
	errState	not a connection listener
	errAborted	request aborted by a dspRemove call

Get Status

This call returns the current state of the connection end specified by the parameter *ccbRefNum*. Note that the values returned in the parameters *sendQPending* and *recvQPending* include any bytes taken up by logical end-of-message indicators.

Parameter block	—>	26	csCode	word	always dspStatus
	—>	32	ccbRefNum	word	refnum of connection end
	<—	34	statusCCB	long	pointer to the CCB
	<—	38	sendQPending	word	bytes waiting to be sent or acknowledged
	<—	40	sendQFree	word	available buffer in bytes of send queue
	<—	42	recvQPending	word	bytes waiting to be read from queue
	<—	44	recvQFree	word	available buffer in bytes of receive queue

Result codes	errRefNum	bad connection refnum

Read Bytes

This call is used by the client to read bytes sent by the remote connection end. The parameter *reqCount* specifies the size of the buffer (in bytes) into which data is to be read. The parameter *actCount* is set to the actual number of bytes read. The parameter *dataPtr* points to a buffer that will receive the bytes.

The call completes when the requested number of bytes have been read or an intervening logical end-of-message indicator is encountered. If the last byte read constitutes the end of a logical message, the parameter *eom* will be set to 1.

If the connection is closed or torn down, outstanding read requests complete in the normal manner. Even though the connection is closed, bytes remaining in the receive queue are still valid data. The client may continue to post read requests to ADSP until there are no more bytes left in the receive queue to be read. The routine Get Status can be called to determine how many bytes remain, or read requests can be posted until the parameters *actCount* and *eom* both return 0. If there are fewer than *reqCount* bytes remaining in the receive queue, the read request completes with *actCount* set to the actual number of bytes being returned.

◆ *Note:* A remote connection end may close the connection immediately after sending a stream of bytes to the connection end. By polling the *state* or *userFlags* fields of the CCB, you may find that the connection has been closed before you have finished reading the bytes from the receive queue. You must decide whether to continue processing the bytes in the receive queue.

Parameter block

—>	26	csCode	word	always dspRead
—>	32	ccbRefNum	word	refnum of connection end
—>	34	reqCount	word	requested number of bytes to be read
<—	36	actCount	word	actual number of bytes read
—>	38	dataPtr	long	pointer to buffer into which bytes are to be read
<—	42	eom	byte	1 if end of message; otherwise, 0

Result codes

errRefNum	bad connection refnum
errFwdReset	read terminated by forward reset
errAborted	request aborted by a dspRemove or dspClose call

Write Bytes

This call is used to send data to the remote connection end. The *refnum* field specifies the connection end from which the data is sent. The parameter *reqCount* specifies the number of bytes to be copied to the send queue, while *actCount* returns the number of bytes that were actually copied. If *reqCount* is 0, no bytes are copied. The parameter *dataPtr* is a pointer to the data to be copied to the send queue.

Setting the *eom* parameter to a nonzero value causes a logical end-of-message indicator to be inserted just after the last data byte to be written. If *reqCount* is 0, only the end-of-message indicator is added to the send queue.

If the *flush* parameter is nonzero, ADSP immediately sends this and any unsent data in the send queue to the remote connection end (assuming there is buffer space at the remote end). If *flush* is 0, the data is placed in the send queue but may not be sent immediately. Details on when data is actually transmitted to the remote connection end can be found in "Set Options" later in this chapter.

Note that bytes written to the send queue are not removed until their receipt has been acknowledged by the remote connection end.

Parameter block	—>	26	csCode	word	always dspWrite
	—>	32	ccbRefNum	word	refnum of connection end
	—>	34	reqCount	word	requested number of bytes to be written
	<—	36	actCount	word	actual number of bytes written
	—>	38	dataPtr	long	pointer to data to be written
	—>	42	eom	byte	1 if end of message; otherwise, 0
	—>	43	flush	byte	1 to send data now; otherwise, 0

Result codes	errRefNum	bad connection refnum
	errState	connection is not open
	errAborted	request aborted by a dspRemove or dspClose call

Send Attention Message

This call is used to send an attention message to the remote connection end. The attention message consists of a 2-byte attention code and up to 570 bytes of attention data.

The parameter *attnCode* is a client-definable code sent in the attention packet. The attention code can contain any value from $0000 to $EFFF inclusive. Attention codes in the range from $F000 to $FFFF inclusive are reserved by ADSP.

The parameter *attnSize* specifies the number of bytes of attention data, and *attnData* is a pointer to the data.

The parameter *attnInterval* specifies the interval between retransmissions in 10-tick (1/6-second) increments. The client can specify any value from 1 (1/6 second) to 180 (30 seconds, the connection probe frequency) inclusive. The attention is retransmitted indefinitely until it is properly acknowledged or the connection fails.

Parameter block					
	—>	26	csCode	word	always dspAttention
	—>	32	ccbRefNum	word	refnum of connection end
	—>	34	attnCode	word	client attention code
	—>	36	attnSize	word	size in bytes of attention data
	—>	38	attnData	long	pointer to attention data
	—>	42	attnInterval	byte	attention retransmit interval

Result codes		
	errRefNum	bad connection refnum
	errState	connection is not open
	errAttention	attention message too long
	errAborted	request aborted by a dspRemove or dspClose call

Set Options

This call allows the client to set options for the connection end specified by the parameter *ccbRefNum*. The send timer defines the frequency of connection-end maintenance by ADSP. At each timer interval, any unsent data bytes in the send queue are flushed. The timer granularity is 10 ticks (1/6 second), and the client can set the *sendTimer* parameter to any value from 1 (1/6 second) to 180 (30 seconds, the connection probe frequency) inclusive. The default interval is 1 (1/6 second). Passing 0 causes the send timer to remain unchanged.

◆ *Note:* There are certain conditions that cause the timer interval to temporarily increase by multiples of itself until it reaches the frequency of the connection probe timer (30 seconds). This behavior is termed *backing off* and typically occurs when there is excessive network traffic. This mechanism prevents the transmission of needless, incessant packets that would further consume network resources. The timer returns to its normal frequency when a packet is received from the remote end.

The send blocking factor allows the client to control when packets are sent based on the number of unsent bytes waiting in the send queue. This feature can be useful in some applications in which the client requests single byte writes. By increasing the parameter *sendBlocking*, the client can reduce network traffic. Given a send blocking factor of *n*, ADSP sends the unsent data bytes only when:

- The number of unsent data bytes is greater than or equal to the send blocking factor *n*.
- The client has requested that all data be flushed in the Write Bytes call.
- Some other event, such as the connection timer expiring, requires that a packet be sent to the remote end so any unsent data bytes will accompany the packet.

The default blocking factor is 16 bytes, but the client can set the parameter *sendBlocking* to any value from 1 to the maximum size of a packet. Setting *sendBlocking* to 0 causes the factor to remain unchanged.

The retransmit timer determines the number of intervals before sent, unacknowledged data in the send queue is retransmitted. The client can adjust the retransmit timer to adapt the connection to network conditions. The AppleTalk Echo Protocol (AEP) can be used to estimate round-trip times and to provide a gauge for setting the retransmit timer. The timer granularity is 10 ticks (1/6 second) and the default interval is 6 (1 second). The client can set the *rtmtTimer* parameter to any value from 1 (1/6 second) to 180 (30 seconds, the connection probe frequency) inclusive. Passing 0 causes the retransmit timer to remain unchanged.

The *badSeqMax* parameter allows the client to set the threshold for sending retransmit advice to the remote end. After receiving some *n* consecutive out-of-sequence packets, it may be more efficient to advise the remote end to retransmit the lost bytes than to wait for the remote end's retransmit timer to expire. Setting *badSeqMax* to 5 causes retransmit advice to be sent to the remote end after 5 consecutive out-of-sequence packets have been received. *badSeqMax* may be set to any value from 1 to 255 inclusive; passing 0 causes the parameter to remain unchanged. The default value is 3.

The parameter *useCheckSum* specifies whether DDP should compute and include a checksum in each packet that is sent to the remote connection end. This feature is enabled only when sending extended-form header DDP packets (internet packets). Regardless of the *useCheckSum* setting, ADSP automatically validates the checksum of any extended-form header DDP packet it receives with nonzero checksum bytes. The default for *useCheckSum* is FALSE.

Parameter block	—>	26	csCode	word	always dspOptions
	—>	32	ccbRefNum	word	refnum of connection end
	—>	34	sendBlocking	word	send blocking threshold
	—>	36	sendTimer	byte	send timer interval
	—>	37	rtmtTimer	byte	retransmit timer interval
	—>	38	badSeqMax	byte	retransmit advice send threshold
	—>	39	useCheckSum	byte	generate DDP checksum on internet packets

Result codes	errRefNum	bad connection refnum

Forward Reset

The forward-reset mechanism allows the client to flush all data written but not yet delivered to the remote connection end's client. The call causes the connection end to reset its send queue and issue a forward-reset control packet to the remote connection end. Upon receipt of the forward-reset control packet, the remote connection end resets its receive queue and informs its client.

The forward reset is nondeterministic, since all the outstanding data may have already been delivered to the remote client.

Parameter block	—>	26	csCode	word	always dspReset
	—>	32	ccbRefNum	word	refnum of connection end

Result codes	errRefNum	bad connection refnum
	errState	connection is not open
	errAborted	request aborted by a dspRemove or dspClose call

Get New CID

This call is useful for clients wanting to open a connection using an alternate means of establishing the two connection ends. The two clients arbitrate the connection-opening parameters using some alternative protocol outside the scope of ADSP. Each client informs the other of the values of its *localCID*, internet socket address, *recvSeq*, *recvWindow*, and *attnRecvSeq*. Each client then calls ADSP to synchronize the two connection ends using the parameters received from the other client.

ADSP clients that want to utilize this connection-opening model should create their connection ends using the Create Connection End call. The Get New CID call then assigns a unique connection identifier to the connection end. This value is returned in the *newCID* parameter so that the client can pass it to the remote client.

Once all open-connection parameters have been determined, each client calls Open Connection, passing *ocEstablish* in the *ocMode* parameter.

Parameter block	—>	26	csCode	word	always dspNewCID
	—>	32	ccbRefNum	word	refnum of connection end
	<—	34	newCID	word	new connection identifier

Result codes	errRefNum	bad connection refnum
	errState	connection end is not closed

Example

In the following example, a client sets up a connection, writes data on it, and then closes the connection.

```
CONST
       qSize  = 600;        {ample space for 512 bytes}
       myDataSize    = 128;        {size of my data writes}

VAR
       error  : OSErr;
       drvrRefNum    : INTEGER;
       connRefNum    : INTEGER;
       dspPB  : DSPParamBlock;
       dspCCB        : TRCCB;
       dspSendQueue  : PACKED ARRAY [1..qSize] OF BYTE;
       dspRecvQueue  : PACKED ARRAY [1..qSize] OF BYTE;
       dspAttnBuffer        : PACKED ARRAY [1..attnBufSize] OF BYTE;
       myData2Write  : PACKED ARRAY [1..myDataSize] OF BYTE;

BEGIN
{make sure .MPP driver is open}
error := MPPOpen;
IF error <> noErr THEN Abort(error);

{open .DSP driver}
error := OpenDriver('.DSP', drvrRefNum);
IF error <> noErr THEN Abort(error);

{create a new connection end}
WITH dspPB DO
       begin
       ioCRefNum := drvrRefNum;
       csCode := dspInit;
       ccbPtr := @dspCCB;
       userRoutine := NIL;
       sendQSize := qSize;
       sendQueue := @dspSendQueue;
       recvQSize := qSize;
       recvQueue := @dspRecvQueue;
       attnPtr := @dspAttnBuffer;
       localSocket := 0;    {dynamically allocate a socket}
       end;
error := PBControl(@dspPB, FALSE);
IF error <> noErr THEN Abort(error);
connRefNum:= dspPB.ccbRefNum;    {save refnum for this connection end}
```

```
{open a connection with a remote end}
WITH dspPB DO
       begin
       ioCRefNum := drvrRefNum;
       csCode := dspOpen;
       ccbRefNum := connRefNum;
       remoteAddress := remAddress;  {probably used NBP to fetch remote address}
       filterAddress := AddrBlock(0);   {open connection with whoever responds}
       ocMode := ocRequest;   {make an Open Connection request}
       ocInterval := 12;   {retry every 2 seconds}
       ocMaximum := 5;   {try 5 times before giving up}
       end;
error := PBControl(@dspPB, FALSE);
IF error <> noErr THEN Abort(error);

{write some data on the open connection}
WITH dspPB DO
       begin
       ioCRefNum := drvrRefNum;
       csCode := dspWrite;
       ccbRefNum := connRefNum;
       reqCount := myDataSize;   {how many bytes to write}
       dataPtr := @myData2Write;   {pointer to data to write}
       eom := 1;   {end-of-message after this data}
       flush := 1;   {send it now, please}
       end;
error := PBControl(@dspPB, FALSE);
IF error <> noErr THEN Abort(error);

{close the connection and remove the connection end}
WITH dspPB DO
       begin
       ioCRefNum := drvrRefNum;
       csCode := dspRemove;
       ccbRefNum := connRefNum;
       abort := 0;
       end;
error := PBControl(@dspPB, FALSE);
IF error <> noErr THEN Abort(error);
END;
```

Appendix **Summary of ADSP data structures**

THIS APPENDIX presents ADSP constants, data types, and assembly language information.

Constants

Driver control ioResults

```
errRefNum       = -1280;  {bad connection refnum}
errAborted      = -1279;  {control call was aborted}
errState        = -1278;  {bad connection state for this operation}
errOpening      = -1277;  {Open Connection request failed or denied}
errAttention    = -1276;  {attention message data too long}
errFwdReset     = -1275;  {read terminated by forward reset}
```

Driver control csCodes

```
dspInit         = 255;    {create a new connection end}
dspRemove       = 254;    {remove a connection end}
dspOpen         = 253;    {open a connection}
dspClose        = 252;    {close a connection}
dspCLInit       = 251;    {create a connection listener}
dspCLRemove     = 250;    {remove a connection listener}
dspCLListen     = 249;    {post a listener request}
dspCLDeny       = 248;    {deny an Open Connection request}
dspStatus       = 247;    {get status of connection end}
dspRead         = 246;    {read data from the connection}
dspWrite        = 245;    {write data on the connection}
dspAttention    = 244;    {send an attention message}
dspOptions      = 243;    {set connection end options}
dspReset        = 242;    {forward reset the connection}
dspNewCID       = 241;    {generate a connection identifier for a
                           connection end}
```

Connection-opening modes

```
ocRequest       = 1;      {request a connection with remote}
ocPassive       = 2;      {wait for a connection request from remote}
ocAccept        = 3;      {accept request as delivered by listener}
ocEstablish     = 4;      {consider connection to be open}
```

Connection end states

```
sListening      = 1;      {for connection listeners}
sPassive        = 2;      {waiting for a connection request from
                           remote}
sOpening        = 3;      {requesting a connection with remote}
sOpen           = 4;      {connection is open}
sClosing        = 5;      {connection is being torn down}
sClosed         = 6;      {connection end state is closed}
```

Client event flags

```
eClosed         = $80;    {received connection-closed advice}
eTearDown       = $40;    {closed due to broken connection}
eAttention      = $20;    {received attention message}
eFwdReset       = $10;    {received forward-reset advice}
```

Miscellaneous constants

```
attnBufSize     = 570;    {size of client attention buffer}
```

Data types

Connection control block

```
TPCCB = ^TRCCB;
TRCCB = PACKED RECORD
    ccbLink          : TPCCB;            {link to next CCB}
    refnum           : INTEGER;          {user reference number}
    state            : INTEGER;          {state of the connection end}
    userFlags        : Byte;             {user flags for unsolicited connection
                                         events}
    localSocket      : Byte;             {socket number of this connection end}
    remoteAddress    : AddrBlock;        {internet address of remote end}
    attnCode         : INTEGER;          {attention code received}
    attnSize         : INTEGER;          {size of received attention data}
    attnPtr          : Ptr;              {pointer to received attention data}
    reserved         : ARRAY [1..220] OF Byte;  {for ADSP internal use only}
    END;
```

Driver control call parameter block

```
DSPPBPtr = ^DSPParamBlock;
DSPParamBlock = PACKED RECORD
    qLink            : QElemPtr;
    qType            : INTEGER;
    ioTrap           : INTEGER;
    ioCmdAddr        : Ptr;
    ioCompletion     : ProcPtr;
    ioResult         : OSErr;
    ioNamePtr        : StringPtr;
    ioVRefNum        : INTEGER;
    ioCRefNum        : INTEGER;          {ADSP driver refnum}
    csCode           : INTEGER;          {ADSP driver control code}
    qStatus          : LONGINT;          {for ADSP internal use only}
    ccbRefNum        : INTEGER;          {refnum of CCB}
    CASE INTEGER OF
    dspInit,
    dspCLInit:
    (
    ccbPtr           : TPCCB;            {pointer to CCB}
    userRoutine      : ProcPtr;          {client routine to call on event}
```

```
sendQSize       : INTEGER;          {size of send queue (0..64K bytes)}
sendQueue       : Ptr;              {client passed send queue buffer}
recvQSize       : INTEGER;          {size of receive queue (0..64K bytes)}
recvQueue       : Ptr;              {client passed receive queue buffer}
attnPtr         : Ptr;              {client passed receive attention buffer}
localSocket     : Byte;             {local socket number}
);

dspOpen,
dspCLListen,
dspCLDeny:
(
localCID        : INTEGER;          {local connection identifier}
remoteCID       : INTEGER;          {remote connection identifier}
remoteAddress   : AddrBlock;        {address of remote end}
filterAddress   : AddrBlock;        {address filter}
sendSeq         : LONGINT;          {local send sequence number}
sendWindow      : INTEGER;          {send window size}
recvSeq         : LONGINT;          {receive sequence number}
attnSendSeq     : LONGINT;          {attention send sequence number}
attnRecvSeq     : LONGINT;          {attention receive sequence number}
ocMode          : Byte;             {open-connection mode}
ocInterval      : Byte;             {Open Connection request retry interval}
ocMaximum       : Byte;             {Open Connection request retry maximum}
);

dspClose,
dspRemove:
(
abort           : Byte;             {abort connection immediately if
                                    nonzero}
);
dspStatus:
(
statusCCB       : TPCCB;            {pointer to CCB}
sendQPending    : INTEGER;          {pending bytes in send queue}
sendQFree       : INTEGER;          {available buffer space in send queue}
recvQPending    : INTEGER;          {pending bytes in receive queue}
recvQFree       : INTEGER;          {available buffer space in receive
                                    queue}
);
```

```
dspRead,
dspWrite:
(
reqCount          : INTEGER;          {requested number of bytes}
actCount          : INTEGER;          {actual number of bytes}
dataPtr           : Ptr;              {pointer to data buffer}
eom               : Byte;             {indicates logical end of message}
flush             : Byte;             {send data now}
);

dspAttention:
(
attnCode          : INTEGER;          {client attention code}
attnSize          : INTEGER;          {size of attention data}
attnData          : Ptr;              {pointer to attention data}
attnInterval      : Byte;             {retransmit timer in 10-tick intervals}
);

dspOptions:
(
sendBlocking      : INTEGER;          {quantum for data packets}
sendTimer         : Byte;             {send timer in 10-tick intervals}
rtmtTimer         : Byte;             {retransmit timer in 10-tick intervals}
badSeqMax         : Byte;             {threshold for sending retransmit
                                      advice}
useCheckSum       : Byte;             {send checksum in extended-form header
                                      DDP packets}
);

dspNewCID:
(
newCID            : INTEGER;          {new connection identifier returned}
);
END;
```

```
      error codes

errRefNum       EQU    -1280   ; bad connection refnum
errAborted      EQU    -1279   ; control call was aborted
errState        EQU    -1278   ; bad connection state for this operation
errOpening      EQU    -1277   ; Open Connection request failed or denied
errAttention    EQU    -1276   ; attention message data too long
errFwdReset     EQU    -1275   ; read terminated by forward reset

   client control codes

dspInit         EQU    255     ; create a new connection end
dspRemove       EQU    254     ; remove a connection end
dspOpen         EQU    253     ; open a connection
dspClose        EQU    252     ; close a connection
dspCLInit       EQU    251     ; create a connection listener
dspCLRemove     EQU    250     ; remove a connection listener
dspCLListen     EQU    249     ; post a listener request
dspCLDeny       EQU    248     ; deny an Open Connection request
dspStatus       EQU    247     ; get status of connection end
dspRead         EQU    246     ; read data from the connection
dspWrite        EQU    245     ; write data on the connection
dspAttention    EQU    244     ; send an attention message
dspOptions      EQU    243     ; set connection end options
dspReset        EQU    242     ; forward reset the connection
dspNewCID       EQU    241     ; generate a connection identifier for a
                               connection end

   open connection modes

ocRequest       EQU    1       ; request a connection with remote
ocPassive       EQU    2       ; wait for a connection request from remote
ocAccept        EQU    3       ; accept request as delivered by listener
ocEstablish     EQU    4       ; consider connection to be open

   connection states

sListening      EQU    1       ; for connection listeners
sPassive        EQU    2       ; waiting for a connection request from remote
sOpening        EQU    3       ; requesting a connection with remote
sOpen           EQU    4       ; connection is open
sClosing        EQU    5       ; connection is being torn down
sClosed         EQU    6       ; connection end state is closed
```

```
   client event flags (bit-mask)

eClosed         EQU    $80      ; received connection-closed advice
eTearDown       EQU    $40      ; closed due to broken connection
eAttention      EQU    $20      ; received attention message
eFwdReset       EQU    $10      ; received forward-reset advice

   miscellaneous equates

attnBufSize     EQU    570      ; size of client attention message

   connection control block equates & size

ccbLink         EQU    0                ; link to next CCB
refnum          EQU    ccbLink+4        ; user reference number
state           EQU    refnum+2         ; state of the connection end
userFlags       EQU    state+2          ; flags for unsolicited connection
                                        events
localSocket     EQU    userFlags+1      ; socket number of this connection end
remoteAddress   EQU    localSocket+1    ; internet address of remote end
attnCode        EQU    remoteAddress+4  ; attention code received
attnSize        EQU    attnCode+2       ; size of received attention data
attnPtr         EQU    attnSize+2       ; pointer to received attention data
ccbSize         EQU    attnPtr+224      ; total byte size of CCB

   adsp queue element equates

csQStatus       EQU    CSParam          ; for ADSP internal use only
csCCBRef        EQU    csQStatus+4      ; refnum of CCB

   dspInit, dspCLInit

csCCBPtr        EQU    csCCBRef+2       ; pointer to CCB
csUserRtn       EQU    csCCBPtr+4       ; client routine to call on event
csSendQSize     EQU    csUserRtn+4      ; size of send queue (0..64K bytes)
csSendQueue     EQU    csSendQSize+2    ; client passed send queue buffer
csRecvQSize     EQU    csSendQueue+4    ; size of receive queue (0..64K bytes)
csRecvQueue     EQU    csRecvQSize+2    ; client passed receive queue buffer
csAttnPtr       EQU    csRecvQueue+4    ; client passed receiving attention
                                        buffer
csLocSkt        EQU    csAttnPtr+4      ; local socket number
```

```
    dspOpen, dspCLListen, dspCLDeny

csLocCID        EQU    csCCBRef+2        ; local connection identifier
csRemCID        EQU    csLocCID+2        ; remote connection identifier
csRemAddr       EQU    csRemCID+2        ; address of remote end
csFltrAddr      EQU    csRemAddr+4       ; address filter
csSendSeq       EQU    csFltrAddr+4      ; local send sequence number
csSendWdw       EQU    csSendSeq+4       ; send window size
csRecvSeq       EQU    csSendWdw+2       ; receive sequence number
csAttnSendSeq   EQU    csRecvSeq+4       ; attention send sequence number
csAttnRecvSeq   EQU    csAttnSendSeq+4   ; attention receive sequence number
csOCMode        EQU    csAttnRecvSeq+4   ; open-connection mode
csOCInterval    EQU    csOCMode+1        ; Open Connection request retry interval
csOCMaximum     EQU    csOCInterval+1    ; Open Connection request retry maximum

    dspClose, dspRemove

csAbort         EQU    csCCBRef+2        ; abort connection immediately if
                                         nonzero

    dspStatus

csSQPending     EQU    csCCBPtr+4        ; pending bytes in send queue
csSQFree        EQU    csSQPending+2     ; available buffer space in send queue
csRQPending     EQU    csSQFree+2        ; pending bytes in receive queue
csRQFree        EQU    csRQPending+2     ; available buffer space in receive
                                         queue

    dspRead, dspWrite

csReqCount      EQU    csCCBRef+2        ; requested number of bytes
csActCount      EQU    csReqCount+2      ; actual number of bytes
csDataPtr       EQU    csActCount+2      ; pointer to data buffer
csEOM           EQU    csDataPtr+4       ; indicates logical end of message
csFlush         EQU    csEOM+1           ; send data now

    dspAttention

csAttnCode      EQU    csCCBRef+2        ; client attention code
csAttnSize      EQU    csAttnCode+2      ; size of attention data
csAttnData      EQU    csAttnSize+2      ; pointer to attention data
csAttnInterval  EQU    csAttnData+4      ; retransmit timer in 10-tick intervals
```

```
   dspOptions
csSendBlocking  EQU    csCCBRef+2        ; quantum for data packets
csSendTimer     EQU    csSendBlocking+2  ; send timer in 10-tick intervals
csRtmtTimer     EQU    csSendTimer+1     ; retransmit timer in 10-tick intervals
csBadSeqMax     EQU    csRtmtTimer+1     ; threshold for sending retransmit
                                         advice
csUseCheckSum   EQU    csBadSeqMax+1     ; use DDP packet checksum
   dspNewCID
csNewCID        EQU    csCCBRef+2        ; new connection identifier returned
```